APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

METHODS AND APPARATUSES FOR REDUCING INFANT MORTALITY IN SEMICONDUCTOR DEVICES UTILIZING STATIC RANDOM ACCESS MEMORY (SRAM)

Inventor(s):
Bryan D. Boatright
Ben J. Eapen
C. Glenn Shirley
Carl Scafidi

Attorney Docket No. 110348-135356
IPG No. P14679

Prepared by:

Express Mail Lab I No. EV370166005US
Date of Deposit: December 31, 2003

METHODS AND APPARATUSES FOR REDUCING BURN IN WITHIN SEMICONDUCTOR DEVICES UTILIZING STATIC RANDOM ACCESS MEMORY (SRAM)

5   BACKGROUND OF THE INVENTION

1.   FIELD OF THE INVENTION

The present invention relates generally to electronic circuits, and more

particularly to methods and apparatuses for reducing burn in.

10   2.   BACKGROUND INFORMATION

Infant mortality is a term used to describe device failures that occur within the

first year of usage. Infant mortality is most often caused by latent manufacturing

defects (hereinafter referred to as "hard defects") that escape detection during factory

tests and become permanently active during use of the device typically rendering the

15   device as non-functional.

Currently, infant mortality in devices is controlled through a pre-shipment process

of burn in. During the burn-in process, devices are operated at artificially elevated

voltages and temperatures for a period of time so as to activate hard defects prior to

device shipment that would otherwise affect customers early in the life of the device.

20   As transistor dimensions and threshold voltages are scaled down for

performance, burn in power and consequently burn in hardware costs increase. This

remains an issue as manufacturers struggle to meet market expectations of infant

mortality rates while maintaining reasonable manufacturing costs.

BRIEF DESCRIPTION OF DRAWINGS

The present invention will be described by way of exemplary embodiments, but not limitations, illustrated in the accompanying drawings in which like references denote similar elements, and in which:

5   Figure 1 is a flow diagram illustrating an overview of the present invention in accordance with one embodiment;

Figure 2 is a flow diagram illustrating a process for detecting hard errors through recurrent ECC events in accordance with one embodiment of the present invention;

Figure 3 is a flow diagram illustrating a process for detecting hard errors through

10   forced ECC events in accordance with another embodiment of the present invention;

Figure 4 is a block diagram illustrating a processor equipped with error correction logic in accordance with one embodiment of the present invention;

Figure 5 is a block diagram illustrating one embodiment of the hard error detection logic of Figure 4;

15   Figure 6 is a block diagram illustrating one embodiment of the cache disable logic of Figure 4;

Figure 7 illustrates an example state diagram that may be implemented via state machine 502 of Figure 5, in accordance with one embodiment of the present invention;

Figure 8 is a block diagram illustrating a processor equipped with error correction

20   logic in accordance with an alternative embodiment of the present invention;

Figure 9 illustrates one embodiment of a generic hardware system for use with various embodiments of the present invention; and

Figure 10 illustrates one embodiment of a machine readable storage medium.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

In accordance with various embodiments of the present invention, a cache-equipped semi-conductor device is provided with enhanced error detection logic to facilitate infant mortality control yielding reduced device burn in time (and cost)

5   accompanied by only a negligible increase in die area. This may be contrasted to prior art cache fault tolerance methods that attempt to detect multiple errors over the full scope of the entire cache (e.g. on a per set basis). Such methods require large amounts of redundant memory cells and circuits, resulting in large increases in die area.

In the following detailed description, numerous specific details are set forth in

10   order to provide a thorough understanding of the present invention. However, those skilled in the art will understand that the present invention may be practiced without these specific details, that the present invention is not limited to the depicted embodiments, and that the present invention may be practiced in a variety of alternative embodiments. In other instances, well known methods, procedures, components, and

15   circuits have not been described in detail.

Various operations will be described as multiple discrete steps performed in turn in a manner that is helpful for understanding the present invention. However, the order of description should not be construed as to imply that these operations are necessarily performed in the order they are presented, or are even order dependent. Furthermore,

20   repeated usage of the phrase "in one embodiment" does not necessarily refer to the same embodiment, although it may. Lastly, the terms "comprising", "including", "having", and the like, as used in the present application, are intended to be synonymous.

Figure 1 is a flow diagram illustrating an overview of the present invention in accordance with one embodiment. In the illustrated embodiment, a cache error is detected within an area of a cache memory at block 102. In one embodiment, detection of the cache error is location-independent in that only a single error need be detected

5   anywhere within the cache memory. In one embodiment, the error may be detected at run-time via error checking and correcting (ECC) logic. A determination is then made at block 104 as to whether the detected error represents a second consecutive error associated with the same area of the cache memory. If not, data identifying the cache area (such as, but not limited to a cacheline) affected by the error is stored at block 106

10   for use in detection of a subsequent error. However, if the detected error represents a second consecutive error associated with the same area of the cache memory, further use of the identified area is prevented at block 108.

In one embodiment, once a cache error has been identified, a cacheline affected by the error may be disabled by a cache management system. For example, a LRU

15   score or MESI state associated with the affected cacheline may be dynamically modified to inhibit further access of the disabled cacheline by e.g. a processor. In one embodiment, a value stored in an LRU register may be modified such that the affected cacheline is less likely to be further accessed by a processor than at least one other cacheline within the cache memory. In one embodiment, a value may be stored in an

20   LRU register associated with the affected cacheline to indicate that the cacheline was most recently used thereby decreasing the probability that the affected cacheline will be accessed further. In another embodiment, a MESI protocol associated with the cache memory may be expanded to include a "disable" state that is assignable to the cacheline.

Using such an expanded disable state may eliminate the probability that a processor will

further access the affected cacheline.

In an alternative embodiment, rather than cache errors being detected at run-time,

errors may be detected at startup via programmable built-in self test (PBIST) logic. In

5   such an embodiment, further use of the affected area may be prevented without

determining whether the error is a second consecutive error associated with the same

area of the cache.

Figure 2 is a flow diagram illustrating a process for detecting hard errors through

10   recurrent ECC events in accordance with one embodiment of the present invention. In

the illustrated embodiment, the process begins at block 202 with a cache access (e.g. by

a processor) resulting in an ECC error. Once an error has been detected, an attempt is

made to determine whether the error is a hard error (e.g. due to a latent manufacturing

defect) or a soft error such as a single event upset (e.g. due to an alpha particle). In

15   Figure 2, a determination is made at block 204 as to whether an error counter is equal to

zero. In one embodiment, the error counter is implemented as a state machine. If the

error counter is equal to zero, an error register may be loaded with the current operation

address at block 207, and the error counter may be incremented or otherwise set to

indicate an ECC error at block 208. Thereafter, the process returns to block 202 where it

20   may wait for additional cache accesses to produce a subsequent ECC error. Upon the

occurrence of a subsequent cache error, a determination may again be made at block

204 as to whether the error counter is equal to zero. If the error counter is not equal to

zero, thus indicating a previous detection of an ECC error, a further determination may

be made at block 205 as to whether the current operation address associated with the

cache access matches an address stored in the error register. If not, the error register

may be loaded with the current operation address and block 207. However, if the current

operation address associated with the cache access matches an address stored in the

5   error register, a cacheline disable function may instead be activated at block 206.

Figure 3 is a flow diagram illustrating a process for detecting hard errors through

forced ECC events in accordance with another embodiment of the present invention. In

the illustrated embodiment, the process begins at block 302, where a determination may

10   be made as to whether a cache access produced an ECC error. Upon a cache access

producing an ECC error, the cacheline that caused the error may be determined at block

303. At block 304, the same cacheline that caused the error may be accessed a second

consecutive time to proactively determine whether the error was a hard error or a soft

error. A determination as to whether the second consecutive cache access also

15   produced an ECC error may further be made at block 306. If the second consecutive

cache access did not produce an ECC error, the process may return to block 302 where

the process waits until another cache access produces an ECC error. However, if the

second consecutive cache access did also produce an ECC error at block 306, the

cacheline affected by the error may be disabled at block 308. In one embodiment, a

20   cache management system such as a LRU algorithm or MESI protocol associated with

the cache may be modified so as to decrease or eliminate the chance of the affected

cacheline being accessed again.

Although the process depicted in Figure 3 may decrease the amount of time it

takes to detect a hard error as compared to the process depicted in Figure 2, a system

equipped to implement the process of Figure 3 may require additional logic/circuitry and

corresponding die area beyond that required by the process of Figure 2.

5

Figure 4 is a block diagram illustrating a processor equipped with error correction

logic in accordance with one embodiment of the present invention.   As shown, processor

400 includes cache memory 402 and error detection block 410. In one embodiment,

cache memory 402 may represent an n-way set associative static random access

10   memory (SRAM) array. Error detection block 410 may include ECC block 414, hard error

detect block 416, and cache disable block 418. ECC block 414 may represent error

checking and correcting circuitry well-known in the art to facilitate detection of soft errors

occurring in cache memory (such as cache memory 402). Since the operation and

configuration of ECC block 414 is well known, it will not be discussed further except with

15   respect to hard error detect block 416 and cache disable block 418.

Hard error detect block 416 represents logic/circuitry designed to detect and

isolate hard errors caused by latent manufacturing defects within cache memory 402. In

one embodiment, hard error detect block 416 may include digital logic and/or analog

circuitry to determine if two consecutive soft errors occur within the same area of cache

20   memory 402. In one embodiment, an error register is used to store data identifying a

set/way area of cache memory 402 affected by an error as reported e.g. by ECC block

414.

Cache disable block 418 represents logic/circuitry designed to prevent an area of

cache memory 402 affected by an error from being accessed any further by processor

400. In one embodiment, cache disable block 418 may disable or otherwise inhibit

further access to a cacheline associated with an ECC error. In one embodiment, access

5   to the affected cacheline may be inhibited via modification of a LRU score associated

with the affected cacheline. Furthermore, access to the affected cacheline may be

disabled via the assignment of an "invalid" state of a modified MESI protocol to the

affected cacheline.

10   Figure 5 is a block diagram illustrating one embodiment of hard error detection

logic 416 of Figure 4. As shown, hard error detection logic 416 includes error register

506 to store data identifying a particular area of cache memory 402 affected by an error

(e.g. as reported by ECC block 414), comparison logic 504 to determine whether the

data stored in error register 506 matches a current operation address, and state machine

15   502 to determine whether a detected error represents a second consecutive error

associated with a common memory location such as a memory address.

During run-time, a current operation address may be provided by processor 400 to

cache 402 and comparison logic 504. If an error is detected in an area of cache memory

402 where the current operation address is stored, ECC block 414 provides an error

20   signal to state machine 502 reporting such error. At substantially the same time, the

current operation address is compared via comparison logic 504 to an address stored in

error register 506 and corresponding to a previous error. If the current operation address

does not match the address stored in error register 506, write enable signal 508 may be

asserted to store the current operation address in error register 506. However, if the

current operation address does match the address stored in error register 506, a cache

disable signal 510 coupled to cache disable block 418 may be asserted.

5   Figure 6 is a block diagram illustrating one embodiment of the cache disable logic

of Figure 4. As shown, cache disable logic 418 includes M-to-N decode logic 602 and

way select logic 612. In one embodiment, M-to-N decode logic 602 generates a way

disable vector based upon received cache disable signal 510. The way disable vector

may be used to select any of (N-1) ways (e.g. via way select lines) as well as modifying

10   a cache management system, such as a LRU score or MESI state assignment

associated with the selected way. In one embodiment, the state of each way select line

is determined based upon the outcome of a corresponding logical AND operation (e.g.

via AND gate 614) performed between an inverted input signal and a non-inverted input

signal for each way select line. In one embodiment, the non-inverted input signal

15   corresponds to the result of a comparison between a cache tag array and the current

operation address (e.g. via cache hit/miss detect logic – not shown), while the inverted

input signal is derived from a corresponding out put signal from decode block 602.

In the embodiment of Figure 6, AND gates 614 are each illustrated as having an

inverted input signal as output signals from decode logic 602 is assumed to be active

20   low. However, other circuit arrangements may be implemented without departing from

the spirit and scope of the present invention as e.g. described with respect to the various

illustrated embodiments. For example, if the output of decode logic is active high, AND

gates 614 may not have an inverted input. In one embodiment, decode block 602

represents a 3-to-8 decoder to facilitate selection and subsequent disabling of any

cacheline of an 8-way set associative cache memory.

Figure 7 illustrates an example state diagram that may be implemented via state

5   machine 502 of Figure 5, in accordance with one embodiment of the present invention.

As shown, the state machine 502 remains idle at state 702 until a first ECC error is

detected via a signal from ECC block 414. Upon such time, state machine 502

progresses to state 704 while error register 506 is assigned the value of the current

operation address such as e.g. the current set/way address. At state 704, state machine

10   502 may pause to allow ECC block 414 enough time to recover (as part of routine ECC

operation) any modified data that may be stored in the affected area of the cache. In

one embodiment state machine 502 may wait to receive an error correct signal from ECC

block 414 indicating that the data has been successfully recovered and the error

corrected. Upon the error being corrected, state machine 502 may proceed to state 706

15   where again it waits for another error to be detected. Upon receiving a second error, a

determination may be made as to whether the set/way address stored in error register

506 is equal to the current address of the error. If not, the error register is assigned the

value of the current set/way address and state machine 502 proceeds back to state 704.

However, if the set/way address stored in error register 506 is equal to the current

20   address of the error, state machine 502 proceeds to state 708 where state machine 502

again may wait for an error correct signal from ECC block 414. Once the error has been

corrected and any modified data recovered by ECC block 414, state machine 502

proceeds to state 710 where the affected area of the cache as e.g. indicated by the stored set/way address is disabled.

Figure 8 is a block diagram illustrating a processor equipped with error correction

5   logic in accordance with an alternative embodiment of the present invention. Processor 800 is similar to processor 400 of Figure 4 with the exception of PBIST block 802. PBIST block 802 represents programmable built-in self test logic /circuitry used to detect errors processor 800, and more specifically in cache 402, as processor 800 proceeds through a startup phase. Upon detecting an error in cache 402, PBIST block 802 may

10   assert a signal causing cache disable block 418 to disable an affected cacheline. Since errors detected by PBIST block 802 are likely hard errors, cache disable block 418 may proceed to disable the affected cacheline and correspondingly modify a cache management system without determining if the error is a second consecutive error associated with a common memory area/address.

15

Figure 9 illustrates one embodiment of a generic hardware system intended to represent a broad category of computer systems such as personal computers, workstations, and/or embedded systems. In the illustrated embodiment, the hardware system includes processor 910 including cache 402 and error detect logic in accordance

20   with embodiments of the invention. Processor 910 is coupled to high speed bus 905, which is coupled to input/output (I/O) bus 915 through bus bridge 930. Temporary memory 920 is coupled to bus 905. Permanent memory 940 is coupled to bus 915. I/O

device(s) 950 is also coupled to bus 915. I/O device(s) 950 may include a display

device, a keyboard, one or more external network interfaces, etc.

Certain embodiments may include additional components, may not require all of

the above components, or may combine one or more components. For instance, cache

5   402 and/or error detect block 410 may be located off-chip with respect to processor 910.

Alternately, permanent memory 940 may be eliminated and temporary memory 920 may

be replaced with an electrically erasable programmable read only memory (EEPROM),

wherein software routines are executed in place from the EEPROM. Some

implementations may employ a single bus, to which all of the components are coupled,

10   or one or more additional buses and bus bridges to which various additional components

can be coupled. Similarly, a variety of alternate internal networks could be used

including, for instance, an internal network based on a high speed system bus with a

memory controller hub and an I/O controller hub. Additional components may include

additional processors, a CD ROM drive, additional memories, and other peripheral

15   components known in the art.

In one embodiment, the present invention, as described above, could be

implemented using one or more hardware systems such as the hardware system of

Figure 9. Where more than one computer is used, the systems can be coupled to

communicate over an external network, such as a local area network (LAN), an internet

20   protocol (IP) network, etc. In one embodiment, the present invention as described above

may be implemented as software routines executed by one or more execution units

within the computer(s). For a given computer, the software routines can be stored on a

storage device, such as permanent memory 940.

Alternately, as shown in Figure 10, the software routines can be machine executable instructions 1010 stored using any machine readable storage medium 1020, such as a diskette, CD-ROM, magnetic tape, digital video or versatile disk (DVD), laser disk, ROM, Flash memory, etc. The series of instructions need not be stored locally, and

5   could be received from a remote storage device, such as a server on a network, a CD ROM device, a floppy disk, etc., through, for instance, I/O device(s) 950 of Figure 9.

From whatever source, the instructions may be copied from the storage device into temporary memory 920 and then accessed and executed by processor 910. In one implementation, these software routines may be implemented in micro-control code. It is

10   to be appreciated, however, that these routines may be implemented in any of a wide variety of programming languages.

In alternate embodiments, the present invention as described above may be implemented in discrete hardware or firmware. For example, one or more application specific integrated circuits (ASICs) could be programmed with one or more of the above

15   described functions of the present invention. In another example, one or more functions of the present invention could be implemented in one or more ASICs on additional circuit boards and the circuit boards could be inserted into the computer(s) described above. In another example, field programmable gate arrays (FPGAs) or static programmable gate arrays (SPGA) could be used to implement one or more functions of the present

20   invention. In yet another example, a combination of hardware and software could be used to implement one or more functions of the present invention.

Epilog

While the present invention has been described in terms of the above-illustrated

embodiments, those skilled in the art will recognize that the invention is not limited to the

embodiments described. The present invention can be practiced with modification and

5   alteration within the spirit and scope of the appended claims. Thus, the description is to

be regarded as illustrative instead of restrictive on the present invention.